REPORT NO: NAWCADPAX--96-195-TR #103

LOW RANK DETERMINATION USING LEAST SQUARES

Peter R. Turner, Ph.D.
Mathematics Department
U S Naval Academy
Annapolis, MD 21402

5 AUGUST 1995

FINAL REPORT
Period Covering June 1995 to August 1995

Approved for Public Release; Distribution is Unlimited.

Prepared for
OFFICE OF NAVAL RESEARCH
800 N. Quincy Street
Arlington, VA 22217

Reviewed By: [illegible] **Date:** 7/9/96
Author/COTR

Reviewed By: [illegible] **Date:** 7/9/96
LEVEL III Manager

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** 5 August 1995	**3. REPORT TYPE AND DATES COVERED** June 1995 to August 1995

4. TITLE AND SUBTITLE

Low Rank Determination Using Least Squares

5. FUNDING NUMBERS

6. AUTHOR(S)

Peter R. Turner, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Air Warfare Center
Aircraft Division Warminster
Code 455100R07
Warminster, PA 18974-0591

8. PERFORMING ORGANIZATION REPORT NUMBER

NAWCADPAX--96-195-TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Mathematics Department
U S Naval Academy
Annapolis, MD 21402

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this report we discuss a technique for determining the rank of a matrix of a special type. The matrix is assumed to be composed of a matrix which has very low rank relative to its magnitude and a noise matrix component. The objective is to determine the rank of the "underlying" matrix. The basic approach explored here is to exploit the observation that the rows of a low rank matrix are linear combinations of a small number of those rows. Therefore if we select "basis" rows carefully, it should be true that the rows of the noisy matrix can be closely approximated by such linear combinations. The approximation is performed easily in a least squares sense and leads to an algorithm which appears to be quite robust and efficient. Its performance is similar in reliability to the use of SVD-based algorithms but with a cost comparable to Gauss elimination or LU factorization.

14. SUBJECT TERMS

Underlying matrix, Low rank, Basis rows.

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

NAWCADPAX--96-195-TR

Low Rank Determination using Least Squares

PETER R TURNER
MATHEMATICS DEPARTMENT, U S NAVAL ACADEMY, ANNAPOLIS MD 21402
PRT@MATH.USNA.NAVY.MIL

ABSTRACT. In this report we discuss a technique for determining the rank of a matrix of a special type. The matrix is assumed to be composed of a matrix which has very low rank relative to its magnitude and a noise matrix component. The objective is to determine the rank of the "underlying" matrix. The basic approach explored here is to exploit the observation that the rows of a low rank matrix are linear combinations of a small number of those rows. Therefore if we select "basis" rows carefully, it should be true that the rows of the noisy matrix can be closely approximated by such linear combinations. The approximation is performed easily in a least squares sense and leads to an algorithm which appears to be quite robust and efficient. Its performance is similar in reliability to the use of SVD-based algorithms but with a cost comparable to Gauss elimination or LU factorization.

1. INTRODUCTION

There are several possible approaches to the problem of determining what we might call the *effective rank* of a (square) matrix. As with many problems where various approaches are possible this allows the user the opportunity to choose between (or, perhaps, balance) reliability or robustness on the one hand and speed or reduced complexity on the other.

First we must explain what we mean by effective rank. Suppose that a matrix A is contaminated with noise or some other source of error so that what we are given is not the original matrix A but some perturbation $\widehat{A} = A + E$ where typically the elements of E are small compared with those of A. The relative scale of the perturbation matrix and the underlying matrix may be better expressed in terms of some norm of A for many situations. Our objective is to determine the rank of A itself.

Clearly the nature of the problem changes if we have some knowledge of the magnitude of the perturbations. This situation often arises in naval applications. (See [1], for example.) It is often the case in important applications that the rank of the underlying matrix A is *very* small relative to its dimension and it is this particular situation that we are interested in here. Rank deficiency of as much as 90% is not uncommon, that is we may have $\text{rank}(A) \approx \dim(A)/10$.

The general problem of rank determination has been studied extensively and has many well-known approaches. Perhaps the most robust is based on the *singular value decomposition* (SVD) of A. The number of singular values greater than a certain threshold value is taken to be the rank of A at that tolerance level. In the absence of noise the threshold is typically set at a level comparable with the machine unit of the floating-point arithmetic system being used. For example, MATLAB[1] uses the default threshold value for its rank function defined for an $n \times n$ matrix by

$$tol := n * s_1 * \mu \tag{1}$$

[1]MATLAB is a registered trademark of The MathWorks, Inc.

where s_1 is the largest singular value of the matrix, and μ is the machine unit of the floating-point system. (See [5] for example.) The singular values are computed by the MATLAB function svd using a maximum of 75 iterations of the QR algorithm. (For details of the underlying algorithms, see [2].)

In the noisy situation, this tolerance can be adjusted to reflect the level of that noise. In (1), the factor $s_1 * \mu$ can be viewed as an absolute error bound on the elements. In that case, it is reasonable that if we know a bound for the noise such that

$$|e_{ij}| < S$$

then we may use

$$tol := n * S$$

in place of (1). This revised tolerance can be justified by viewing the noise matrix as an initial error matrix which dominates the effect of the floating-point errors. In our experiments this proved to be a satisfactorily reliable tolerance level. A statistical study of how the tolerance level for the svd algorithm should be set in practice was undertaken in [4].

The problem with this SVD-based rank-determination algorithm is that it is expensive to compute. It, of course, provides much more information than is being sought here. If this more detailed knowledge of the matrix is required for subsequent parts of the problem then it would probably remain a likely choice. However, in our low-rank setting solving the complete SVD solely in order to identify the rank is often too expensive.

There is another potential difficulty with the use of the SVD arising out of the interpretation of the results. Suppose, for example, the original matrix has rank 4 but that one of its singular values is very small — of the same order of magnitude as the noise perhaps. The question is now how to interpret this in the presence of noise. This is really a question of conditioning (or perhaps a definition of ill-conditioning) for this particular problem. The assumption made here is that any singular values which are below tolerance are the result of noise and therefore that they should rightly be neglected in computing the rank. The effective rank in our example would be 3. In our experiments this situation *never* arose in practice.

At the other end of the computational cost scale is the use of Gauss elimination or LU factorization (with pivoting). Typically, our perturbed low-rank matrix will have full rank, but the diagonal entries of the resulting upper triangular factor may still be expected to be small where exact zeros would be obtained for the exact matrix using exact arithmetic. Again, by regarding the noise as an initial error matrix we could set a tolerance for these diagonal entries of $n * S$. However as was seen in the experiments discussed in [6] the resulting diagonal entries can be of similar magnitude to some of those that would arise from the underlying matrix. It follows that correct determination of the effective rank using Gauss elimination alone is likely to be unreliable. However, the LU factorization is much cheaper to compute than the SVD. By repeating the factorization for several further perturbations of the matrix, it is possible in most cases to get good results. (Indeed, the original factorization yields the correct result most of the time.)

Our objective here is to obtain a sufficiently reliable technique for determining the rank of a low-rank matrix whose cost is comparable with that of LU factorization. This technique is based on the last squares approximation of rows of the matrix with linear combinations of other rows. The basic idea behind the algorithm is described in the next section. In Section 3, we consider an improved solution to the least squares problems (which are of low dimension) which reduces to using a rearrangement of the Gram-Schmidt process for orthogonalizing the rows of the matrix. In Section 4, the results of some experiments are presented to support the claim that this algorithm gives reliable results and to examine its computational costs. The results are summarized in the final section of the report.

2. Least squares: basic idea

The basic idea behind this least squares approach to obtaining the effective rank of a low-rank matrix has two aspects. The first is that we should be able to test for ranks 1, 2, and so on separately without the need for a complete SVD or eigenvalue analysis of the matrix. The statistical method based on the coefficients of the characteristic polynomial described in [1] requires the computation of both the complete determinant and several principal subdeterminants in order to test for rank deficiency by first testing for full rank and then for rank $n-1, n-2, \ldots$. This is inevitably more costly than we would like in the event of low rank.

Initially, consider the situation of an $n \times n$ matrix A whose true rank is just 1. Then it follows that every row of this matrix is just a scalar multiple of the first (nonzero) row. In the case of a noisy matrix given by

$$\widehat{A} = A + E \tag{2}$$

then we would expect the same condition to be satisfied *approximately.* This leads to a simple idea for testing for $\widehat{A}$ having effective rank 1 when we know a bound on the elements of E. The idea is that we approximate each row of the matrix by a scalar multiple of the largest row in the sense of least squares approximation and subtract this rank 1 matrix from the original. If the result is small enough, we conclude that the underlying matrix had (effective) rank 1. This is summarized in the following simplified form of the algorithm.

Algorithm 1 Simplified rank 1 test

```
Input
    Contaminated $n \times n$ matrix $\widehat{A}$ with rows denoted by $A(i,:)$
    Bound $S$ on the elements of the noise matrix, and a tolerance level $tol$.
Compute
    Find $p$ such that $||A(p,:)||_1 = \sum_j |a_{pj}|$ is maximum
    for $i = 1$ to $n$
        $B(i,:) := A(i,:) - \frac{A(i,:)\bullet A(p,:)}{A(p,:)\bullet A(p,:)} A(p,:)$
Test
    if $\max(||B(i,:)||_1) < tol$ then effective rank is 1.
```

Remark 1. *Note that the least squares approximation of one vector* $\mathbf{b}$ *by another* $\mathbf{a}$ *of the same dimension is* $\alpha\mathbf{a}$ *where* α *is chosen to minimize* $||\mathbf{b} - \alpha\mathbf{a}||_2^2$ *and this minimum is given by* $\alpha = \frac{\mathbf{a}\bullet\mathbf{b}}{\mathbf{a}\bullet\mathbf{a}}$. *This is the approximation used in Algorithm 1.*

The key questions which remain are how to extend this beyond testing for effective rank 1 and how to set the tolerance level which clearly must depend in some manner on S and n. The issue of the tolerance will be discussed later.

The first of these questions has a simple answer in principle. If it is determined that the original matrix is of rank at least r then we simply approximate (in the least squares sense) all rows by linear combinations of a choice of r rows. Clearly this begs a further question: how do we choose the "right" r rows? This question has a fairly natural answer which emerges from consideration of the least squares problem. Before discussing this in detail, we note that in the situation of low rank that we are considering, the dimension of the normal equations for the various least squares problems is small and therefore inexpensive to solve.

The least squares approximation of a vector $\mathbf{c}$ by a linear combination of vectors $\mathbf{a}, \mathbf{b}$ is obtained by minimizing $||\mathbf{c} - \alpha\mathbf{a} - \beta\mathbf{b}||^2$ and this solution is given by the 2×2 system

$$\begin{bmatrix} \mathbf{a}\bullet\mathbf{a} & \mathbf{a}\bullet\mathbf{b} \\ \mathbf{a}\bullet\mathbf{b} & \mathbf{b}\bullet\mathbf{b} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \mathbf{a}\bullet\mathbf{c} \\ \mathbf{b}\bullet\mathbf{c} \end{bmatrix} \tag{3}$$

This generalizes in the obvious way so that the least squares approximation of a vector $\mathbf{b}$ by a linear combination of the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r$ is $\sum \alpha_j \mathbf{a}_j$ where the coefficients are given by the normal equations

$$\begin{bmatrix} \mathbf{a}_1 \bullet \mathbf{a}_1 & \mathbf{a}_1 \bullet \mathbf{a}_2 & \cdots & \mathbf{a}_1 \bullet \mathbf{a}_r \\ \mathbf{a}_2 \bullet \mathbf{a}_1 & \mathbf{a}_2 \bullet \mathbf{a}_2 & \cdots & \mathbf{a}_2 \bullet \mathbf{a}_r \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{a}_r \bullet \mathbf{a}_1 & \mathbf{a}_r \bullet \mathbf{a}_2 & \cdots & \mathbf{a}_r \bullet \mathbf{a}_r \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_r \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \bullet \mathbf{b} \\ \mathbf{a}_2 \bullet \mathbf{b} \\ \vdots \\ \mathbf{a}_r \bullet \mathbf{b} \end{bmatrix} \tag{4}$$

Subject to choosing the appropriate tolerance level, solving such systems allows us to extend Algorithm 1 to the problem of determining the effective rank by testing for ranks 1, 2, ... in turn. However this would become a potentially expensive process as the rank increases. We shall see in the next section that we can use the earlier results fruitfully to reduce this load.

We turn now to the question of choosing the tolerance. Suppose that the underlying matrix A has true rank 1 and consider the matrix B generated by Algorithm 1. The choice of what we might term a "pivot" row in Algorithm 1 is based on the 1-norm for simplicity of the calculation. It is not necessarily true therefore that $A(p,:)$ has the largest 2-norm. Nonetheless it is likely that the multipliers $\frac{A(i,:)\bullet A(p,:)}{A(p,:)\bullet A(p,:)}$ are bounded by 1 and therefore that the elements of B would be bounded by $2S$. The corresponding bound for a rank 2 test where each row is modified by subtracting the appropriate combinations given by solving systems like (3) would be $3S$ and, in general, for rank r the bound would be $(r+1)S$.

If we assume that the noise level is small compared with the underlying matrix, then using $tol = nS$ would likely be safe throughout. Similarly for a noise matrix whose entries are taken from the normal distribution $N(0, \sigma^2)$ taking $tol = 2\sqrt{n}\sigma$ is probably suitable. (The factor of 2 comes from the fact that approximately 95% of a normal distribution lies within 2 standard deviations of its mean. The $\sqrt{n}$ reflects the growth of the standard deviation in the sum of normal distributions. See [3].) As a general rule therefore it seems that using either nS or $n\sigma$ should be an acceptable tolerance.

We shall see in the next section that the modified version of the algorithm will generate a greater potential rate of growth in the modified noise matrix. We shall therefore revisit this issue there.

3. Orthogonality - rearranged Gram-Schmidt

In this section, we describe a better method for solving the least squares problem which reduces in this case to a particularly simple algorithm. The resulting algorithm turns out to be essentially equivalent to the beginning of a rearrangement of the Gram-Schmidt orthogonalization process. See [2], for example. This rearrangement is not the well-known modified Gram-Schmidt algorithm. It is also important to note that in the low-rank case it is equivalent to only a small part of the full orthogonalization.

Observe first that the solution of (3) or (4) would be trivial if the vectors $\mathbf{a}, \mathbf{b}$ or $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r$ were orthogonal since the resulting matrices are then just diagonal. The complete orthogonal factorization of the original matrix is essentially the equivalent of performing the SVD using QR factorization. This is the expensive approach which we wish to improve upon.

A second observation is that the central operation in Algorithm 1 (that is the operation inside the for loop) is replacing the i-th row of the matrix by its component orthogonal to the "pivot" row $A(p,:)$. The complete loop is therefore replacing every row with a vector orthogonal to this pivot row. This loop is therefore equivalent to the first step of a rearranged Gram-Schmidt process in which the component in the direction of the first vector is removed from *all* the others before going on to obtain further members of the orthogonal basis. We now have all the components of our final algorithm.

The matrix B generated by Algorithm 1 has (at least) one row consisting entirely of zeros and all other rows are orthogonal to $A(p,:)$. The row of the original matrix which is "most orthogonal" to $A(p,:)$ is the "largest" row of B. If the test in Algorithm 1 fails, it can therefore be reapplied to B to test it for rank 1 which would be equivalent to the original matrix having effective rank 2. This process can be repeated until the test is satisfied. The effective rank is then just the number of times Algorithm 1 has been used.

The orthogonalization of the normal equations is being performed on-the-fly so that each iteration is very simple. In the situation where the rank is very low, it is apparent that much of the work which would be required for a complete orthogonal factorization has been avoided.

As just described, the algorithm would be slightly wasteful since the pivot row would be replaced by *exact* zeros on each iteration. Clearly it should be omitted from subsequent iterations. This can most easily be achieved either by performing a row interchange and modifying the limits on the basic loop, or by storing a list of rows which have been used just as a permutation vector is used to avoid interchanges in Gauss elimination.

In the following algorithm, we use row-interchanges since that simplifies the description.

Algorithm 2 Least squares low rank determination

```
Input
    Contaminated $n \times n$ matrix $A$ with rows denoted by $A(i,:)$
    Bound $S$ on the elements of the noise matrix
Initialize
```

```
  $r = 0$
Repeat
  $r := r + 1$
  Test for rank $r$
    Find $p \geq r$ such that $||A(p,:)||_1 = \sum_j |a_{pj}|$ is maximum
    Interchange rows $p, r$
    for $i = r + 1$ to $n$
      $A(i,:) := A(i,:) - \frac{A(i,:) \bullet A(p,:)}{A(p,:) \bullet A(p,:)} A(p,:)$
    $A(r,:) := 0$
    Define a 0-1 (Boolean) test matrix:
      $tst := [abs(A) > n * S]$
  until $||tst||_\infty < n/10$
Output
  $A$ has effective rank $r$.
```

Remark 2. *The use of $n/10$ in the final test here is intended to allow for some components not being "zeroed" as a result of accumulation of roundoff error during the calculation. In the experiments reported in the next section, we never encountered a case where using $||tst||_\infty = 0$ would have produced a different result. Note that this test means that every row of the final matrix has at least 90% of its entries below the threshold.*

Typically we find in the experiments that the use of this algorithm is comparable in accuracy with using the svd-based approach but at a cost (measured in terms of numbers of floating-point operations) which is considerably less. Because Algorithm 2 is iterative in nature it is obviously even more economical when the rank is very low. However even on matrices of full rank it was approximately 50% cheaper than the svd-based algorithm. In the case of a 30×30 matrix of rank just 2, the saving was over 90% with 96K flops for the svd being reduced to just 7030 for Algorithm 2.

The MATLAB code for implementing the algorithm and the experiments will be discussed in the next section.

4. Experimental evidence

In this section we describe some preliminary computational experiments which give reason to believe that the least squares based algorithm for low-rank detection merits further investigation. These experiments were carried out using MATLAB.

The first task is to generate test matrices which are "noisy" versions of matrices of specified rank and dimension. The noise level is also a parameter that can be adjusted. This is done by first generating a random matrix with the specified rank and then perturbing it by the addition of a noise matrix. In the preliminary experiments the noise matrix was what might be termed *absolute noise* as opposed to relative noise. That is once a noise level s, say, is specified, the elements of the noise matrix are chosen to be random within the range $[-s, s]$. Relative noise would correspond to perturbing each element of the exact matrix by a random factor in some small interval around 1. The elements of the noise matrix could also be chosen to come from any given statistical distribution. In these experiments they were always taken from a uniform distribution. MATLAB also has a normal distribution random number generator which delivers random values from a normal (or Gaussian) distribution with a specified standard deviation centered at 0.

All of these variations on the basic theme of uniformly distributed absolute noise will be tested subsequently. Since over 95% of a normal distribution lies within 2 standard deviations of its mean, there is no intuitive reason to believe that the use of Gaussian noise would seriously affect the performance of any particular algorithm or the relative performance of possible choices. It may have an effect on the choice of the tolerance level. Within the context of the svd-based rank-detection algorithm this was examined in [4].

The algorithm used here for generating our test matrices used two simple MATLAB m-files. The first generates a random matrix of specified rank r and dimension $n \times n$ by first generating a complete random matrix of the appropriate dimension and then using random linear combinations of its first r rows to generate the reduced rank matrix. Again the random matrix and the coefficients for the linear combinations were chosen to be uniformly distributed within specified ranges. Specifically the initial $n \times n$ matrix had elements in $[-10, 10]$ and the coefficients for the linear combinations were in $[-3, 3]$. It follows that absolute bounds on the elements of the exact rank r matrix are $[-30r, 30r]$. In practice, of course these elements are likely to be in a much smaller interval than this suggests. There is the theoretical possibility that the resulting matrix may have rank less than r; but this never happened in practice. The second of the m-files generates an $n \times n$ noise matrix with entries uniformly distributed in $[-s, s]$. The final test matrix is obtained by adding these two matrices together to obtain a matrix whose true rank is likely to be n but whose effective rank should be r.

These m-files and the generation of a test matrix are as follows.

```
% Generate new matrix of specified size n and rank r

function A = testmat(n,r);
 a=20*rand(n)-10;
 A=zeros(n);
 for i=1:n
   for j=1:r
      A(i,:)=A(i,:)+(6*rand-3)*a(j,:);
   end
end
 % Generate a noise matrix with "absolute noise"
% Parameters are dimension, noise level

function E=noisemat(n,s);
 E=s*(2*rand(n)-1);

Ahat=testmat(n,r)+noisemat(n,s);
```

In all cases, the least squares based approach was tested against both the svd-based algorithm and the use of the LU factorization (or Gauss elimination). This comparison was based on both the accuracy of the results and the computational complexity of the different algorithms as measured by MATLAB's built-in function flops which gives a count of the total number of floating-point operations used in the current session. Subtraction of values of flops at various stages gives an accurate count of the work involved in performing any one algorithm.

The least squares rank estimation algorithm is implemented in the following m-file which in turn calls the rank-one testing algorithm listed below.

```
% Rank test for noisy matrix using least squares fitting

% Tests rank iteratively using
%       rtest.m
% to perform least squares fit of rows by largest row

% Inputs are matrix and the (estimated) noise level
% Output is its rank as determined by this algorithm

function r=lsrank(A,s);
 n=size(A,1);
 tst=ones(n);
 done=norm(tst,inf)<n/10;
 r=0;
 while ~done
   r=r+1;
   [A,tst]=rtest(A,n,s,r);
   done=norm(tst,inf)<n/10;
 end;
% End of iteration
```

This m-file implements the outer iteration of Algorithm 2. The least squares fitting of each row by the largest row is performed by the following code which modifies the current matrix by subtracting from each “active” row its least squares approximation as a scalar multiple of the largest row. Finally, the Boolean test matrix is generated for use in the rank determination step of the (above) calling program.

```
% Rank 1 test iteration including row interchange
% using least squares approximation of rows

% Outputs are a "modified" matrix B and a test matrix
% which indicates the number of elements in
% each row of B which have not been "zeroed"

% Inputs are matrix A, its dimension n, noise level s
% and the current rank being tested
%
function [B,test] = Rtest(A,n,s,rank);
 B=A;
 for i=1:n
```

```
  zerow(i)=0;
end;

for i=rank:n
  L1(i)=norm(B(i,:),1);
end
[L,p]=max(L1(rank:n));
p=p+rank-1;       % Be careful max counts from beginning of its
                  % argument array NOT the parent array

% Now interchange the rows
 tmp=B(p,:);
 B(p,:)=B(rank,:);
 B(rank,:)=tmp;

 Bt=B';

 for j=rank:n
   sp(j)=B(rank,:)*Bt(:,j);
 end
 for j=rank+1:n
   b(j)=sp(j)/sp(rank);
   B(j,:)=B(j,:)-b(j)*B(rank,:);
 end
 B(rank,:)=zerow;
 L1(rank)=0;
 test=abs(B)>n*s;
% End of iteration
% Max row sum of test < tolerance indicates "success"
```

In the early stages of the testing of this algorithm, the computation was performed interactively in the MATLAB command window. This allowed its progress to be monitored and led to some interesting observations — including the discovery of one particular 20×20 matrix of effective rank 4 where the use of the LU factorization failed to yield good results even when it was used repeatedly to try a statistical approach to the problem. We shall discuss this particular example shortly.

The most critical aspect of this algorithm is setting the tolerance level within the rank one test. This has much greater effect on the reliability of the results than the tolerance level of the outer algorithm. This aspect has only been explored through limited experiments. The values $n \times s$ used in the above m-file is not necessarily the best value — indeed the experiments suggest some modification of this level. This is easily achieved in experiment by modifying the call to the m-file lsrank using something other than the noise bound s.

The first experiments were conducted using the 7×7 matrix of rank 4 which formed the basis of the statistical analysis in [4]. This matrix is

$$A = \begin{bmatrix} 3 & 2 & 1 & 7 & 4 & 5 & 3 \\ 1 & 4 & 2 & 6 & 5 & 10 & 3 \\ 8 & 1 & 5 & 13 & 5 & 7 & 0 \\ 4 & 2 & 7 & 15 & 11 & 11 & 4 \\ 1 & 2 & 1 & 3 & 2 & 5 & 1 \\ 2 & 1 & 3 & 5 & 3 & 5 & 0 \\ 3 & 10 & 1 & 5 & 2 & 21 & 1 \end{bmatrix}$$

to which a noise matrix with random entries can be added in order to conduct experiments.

The first group of tests used $s = 0.1, 0.3$ and 0.5 which are grouped around the critical noise levels in terms of correct evaluation of the effective rank. The least squares algorithm was compared with the svd-based algorithm for a small number of different perturbations of the matrix A. The comparisons were based both on the accuracy of the results and on the number of floating-point operations used to compute them.

For a given matrix dimension and a particular (reported) effective rank, the number of floating-point operations used will be constant for the least squares algorithm. This is not necessarily true of the svd although the variation is likely to be relatively small for different perturbations of the same matrix irrespective of the reported rank. A complete complexity analysis of the least squares based algorithm can be performed and will be reported at a later stage. For the present we content ourselves with the operation counts reported by MATLAB using its built-in function flops.

The results of this first batch of tests are summarized in Table 1. It is immediately apparent that the least squares algorithm represents a substantial saving in work and that for $s = 0.1, 0.5$ its results are equally reliable. For $s = 0.3$ the saving of work is still apparent but the effective rank is consistently reported as 3 rather than 4. This prompted the further experiments on setting the appropriate tolerance level.

TABLE 1 Comparison of least-squares and svd-based algorithm for different perturbations of the 7×7 matrix A of [4]

The tolerance levels used were $n \times s$ for both algorithms

s	Least squares		svd algorithm	
	r	flops used	r	flops used
0.1	4	603	4	2233
	4	603	4	2285
	4	603	4	2233
0.3	3	496	4	2234
	3	496	4	2232
	3	496	4	2285
0.5	3	496	3	2233
	3	496	3	2333
	3	496	3	2334

The savings even for a relatively small matrix with rank which is not *very* small relative to its dimension shows the potential saving to be made by using the least-squares based algorithm. However, around this critical noise level there is clearly some difficulty with identifying the correct effective rank — with this tolerance level.

The next stage of the computational experiment was therefore to try adjusting this to improve the reliability of the least squares algorithm. The saving in computational effort will not be affected by changing the tolerance and so this phase of the testing was restricted to the least squares algorithm. The fact that all the failures reported too low a rank, suggested that the ceiling of ns for elements of the residual matrix was too easily achieved. The experiments were therefore based on finding a lower ceiling to use in defining test in the function rtest above.

First ten tests were run using $s = 0.3$ with the tolerance set as s rather than the ns used previously. This resulted in $r = 4$ nine times and $r = 5$ once. This clear improvement suggested however that maybe the tolerance had been adjusted too much.

The remaining tests using this underlying matrix used instead the tolerance $\sqrt{n} \times s$ as a compromise between the two levels tried hitherto. With $s = 0.3$, 10 out of 10 trials resulted in $r = 4$. The same outcome was obtained with $s = 0.5$ suggesting that a much more reliable tolerance level had been found. For $s = 1.0$ the reliability again fell off with $r = 3$ being returned 6 times out of 10 trials. However, we should observe here that $s = 1$ is of a similar order of magnitude to many elements of the original matrix so that we might expect some further loss of rank to result from such changes. For example with changes to individual entries no greater than ± 1, the fifth and sixth rows of A can be made identical. For the svd-based algorithm $r = 3$ resulted only 3 times — but the other seven all gave $r = 2$ — although, maybe, similar experiments with the tolerance for that algorithm would have yielded better results. This conjecture was tested quite extensively using teh built-in svd function in MATLAB. These tests used normally distributed perturbations of the matrix elements and tests based on the standard deviation of these. Many runs were performed for different noise levels and different tolerance settings. Setting the tolerance at $n\sigma$ was almost universally successful for all noise levels tested whereas slightly higher or slightly lower settings resulted in much less relaible results.

To see whether this reduced tolerance level was likely to prove reliable with our least squares based algorithm, it was further tested for smaller noise-levels $s = 0.01$ and $s = 0.0001$. In both cases, perfect results were obtained.

The primary purpose of our initial experiments was not to examine the tolerance level but to test the feasibility of the least squares approach. In order to test that further experiments with larger matrices were conducted. In most situations (unless the noise approaches the critical level) the sensitivity to the tolerance appears to be low and so the algorithm as originally described was used for most of this testing. The extreme case is that an effective rank of 1 should be identified correctly and reliably. This was tested for various examples with complete success. Testing teh rank one case was also a necessary part of checking teh code for higher rank testing.

For example, two 6×6 matrices of effective rank 1 were tested using the least squares, LU factorization and svd-based algorithms. In all cases the correct effective rank was obtained. The numbers of floating-point operations required were 150 for least squares, 130 for the LU factorization and about 1650 for the singular value decomposition. Even for a very low dimensional example such as this the savings are obvious. (Note that the cost of either least squares or LU rank determination here is comparable with that of setting up the problem in the first place.) In all these rank 1 cases, the noise level was small so that the correct identification of the effective rank was to be expected — but these examples did provide some confirmation of the correctness of the code and the practicality of the approach.

Once this first hurdle had been cleared and the tests discussed above using the test matrix of [4] had been completed further preliminary experiments were conducted with matrices of higher dimension and with varying effective ranks. This testing justifies the claim that this algorithm is worthy of further investigation — both analytic and computational.

One of the first tests performed here was particularly interesting. With $n = 20, r = 4$ and $s = 0.5$ a random test matrix was generated which had interesting properties. The svd algorithm with tolerance ns gave the correct result $r = 4$ using 30722 floating-point operations. The MATLAB LU factorization routine (with partial pivoting) generates an upper triangular factor whose diagonal elements can be used to estimate the rank by counting the number of them greater than some tolerance. Again ns seems a reasonable tolerance to use. For this same matrix, this algorithm gave $r = 2$ — but it was much cheaper of course. The LU factorization used just 5151 flops which represents a saving of over 83%. Before discussing the performance of the least squares algorithm on this example it is worth looking at the singular values and the diagonal of U.

The singular values were reported as (approximately) 247.5, 219.1, 145.3, 110.3, 2.33, 1.96, 1.73, 1.55, 1.52, 1.34, 1.25, 0.99, 0.92, 0.67, 0.50, 0.47, 0.35, 0.25, 0.11 and 0.0001 which show a very marked drop after the first 4. Also almost any likely tolerance would be likely to fall in this gap. (Note however that $\sqrt{ns}$ would *not*.)

The diagonal of U in decreasing absolute value are 39.4, 31.4, 7.0, 6.8, 5.7, 4.5, 2.42, 2.19, 2.11, 1.24, 1.20, 1.05, 0.98, 0.84, 0.82, 0.77, 0.62, 0.61, 0.56 and 0.001. It is almost impossible to envisage any tolerance level which could separate these in such a way that we would have any confidence in the results. The largest relative changes in these magnitudes, for example, would probably result in the conclusion that the effective rank was 2, or 6, or 9, or 19 — none of which is the correct result of course.

This same example was given to the least squares algorithm in an interactive way so that we can see how the algorithm is progressing. After each iteration, the number of entries in each row which were below the tolerance ns was tabulated. Listed as row vectors these were:

Iteration 1: 8, 17, 8, 9, 7, 20, 18, 8, 18, 11, 8, 9, 12, 6, 8, 12, 12, 17, 11, 6
Iteration 2: 14, 17, 14, 9, 16, 20, 17, 12, 20, 11, 13, 19, 10, 19, 17, 13, 20, 20, 12, 20
Iteration 3: 16, 20, 20, 20, 20, 20, 19, 20, 20, 20, 13, 19, 13, 20, 20, 12, 20, 20, 12, 20
Iteration 4: 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20

These results leave little doubt as to the correct conclusion that the effective rank is indeed 4. The cost of this computation (which in this interactive use was not especially efficient) was just 8005 flops representing a saving of 74% relative to the effort required for the singular value decomposition — but this time, unlike the LU factorization, with the same correct result.

The m-files listed above take account of rows which have already been used as the "pivot" row in the approximation and so reduces the number of rows still being worked on. It can be seen from the above example that further economy could be obtained by omitting any rows which have been effectively zeroed from subsequent stages. Even without that further saving but just using the above m-files, the operation count for this example is reduced to just 5855 which is not much greater than the count for the LU factorization. The saving of over 80% in computational effort is certainly encouraging.

The next tests were again with $n = 20$ but with a higher noise level $s = 1$ and an even greater rank deficiency, $r = 3$. The results were again perfect in the sense that the least squares algorithm returned the same correct answer as the svd in each case. The operation count was 4513 for least squares and averaged about 32070 for svd. As the noise level was turned up, it again became apparent that the tolerance needs to be adjusted. This is consistent with the results described above. The conclusions are also consistent with those above. For example, with $n = 20, r = 3, s = 5$, using tolerance ns yielded rank 1 while reducing it to $ns/5 \simeq \sqrt{n}s$ gave the correct result.

Testing this for higher rank and lower noise levels, it appeared that the results were not highly sensitive to the exact tolerance. For example with $n = 20, r = 6, s = 0.01$ tolerances of $20s$ and $2s$ were both good while s gave rank 13. (For comparison, the svd based algorithm with $20s$ gave the expected rank = 6, while a tolerance of just s gave rank = 15.) The operation count saving was of course not as great since more iterations of the least squares algorithm are needed. The least squares algorithm used 8296 flops compared with approximately 30200 for the svd which is still a saving of some 72%. Even with high rank matrices $r = n = 20$ the least squares algorithm saved approximately 50% of the computation while returning the same results as the svd. The pattern of these results was maintained for limited tests of even larger dimension and with varying ranks and noise levels.

The tests were insufficiently extensive to merit detailed report at this stage but they provide further evidence for the desirability of further testing and analysis.

5. Conclusions and suggestions for further work

This report describes a possible algorithm for rank detection based on least squares approximation. This algorithm can be viewed as a rearrangement of the Gram-Schmidt orthogonalization process. The algorithm has been subjected to limited preliminary testing and some analysis which suggests that it is worthy of more detailed study.

Specifically, the algorithm implementation can (and should) be further improved to take advantage of any rows which have been effectively zeroed. The tolerance level used should be both analysed and subjected to experimental testing. The computational complexity analysis of the algorithm should be completed and reported so that an accurate measure of its cost can be obtained. The computational experimentation should be extended to a systematic study of its performance and how this performance compares with well-implemented alternatives. The possible advantages to be gained from possible parallel computer implementation of this algorithm should also be investigated. The parallelism in the new algorithm should be easily exploited whereas the svd is much less obviously parallelizable.

Overall, the new algorithm merits serious study. This study should be undertaken as soon as possible given the naval importance of the problem of low-rank detection which is where the gains to be made from this algorithm are at their greatest.

References

[1] R.Gleeson and Robert M Williams, *NAWCAD report on c_k approach*, NAWCAD Tech Report

[2] G.Golub and C.F.van Loan, *Matrix Computations, 2nd edition*, Johns Hopkins University Press, 1989

[3] R.V.Hogg and A.T.Craig, *Introduction to Mathematical Statistics*, 5th Ed, Prentice Hall, 1995

[4] K.Konstantinides and Kung Yao, *Statistical analysis of effective singular values in matrix rank determination*, IEEE Trans ASSP 36 (1988) 757-763

[5] The MathWorks, Inc. *The Student Edition of MATLAB, User's guide*, Prentice Hall, 1995.

[6] P.R.Turner, *Gauss Elimination: Workhorse of Linear Algebra*, NAWCAD Tech Rep, NAWCADPAX 96-194-TR, 1996

Distribution List

Report No. NAWCADPAX- -96 - 195 - TR

	No. of Copies
Office of Naval Research	2
800 N. Quincy St.	
Arlington, VA 22217	
Marine Corps Research Center	
2040 Broadway Street	
Quantico, VA 22134-5107	
Marine Corps University Libraries	2
Naval Air Systems Command	
Air-5002	
Washington, DC 20641-5004	
Technical Information & Reference Center	2
Naval Air Warfare Center, Aircraft Division	
Building 407	
Patuxent River, MD 20670-5407	
Naval Air Station Central Library	2
Naval Air Systems Command (NAVAIR)	
Jefferson Plaza Bldg 1., 1421 Jefferson Davis Hwy	
Arlington, VA 2243-5120	
Director Science & Technology (4.0T)	2
Naval Sea Systems Command	
2531 Jefferson Davis Hwy	
Arlington, VA 22242-5100	
Technical Library, (SEA04TD2L)	2
Defense Technical Information Center	
Cameron Station BG5	
Alexandria, VA 22304-6145	
DTIC-FDAB	2
U.S. Naval Academy	
Annapolis, MD 21402-5029	
Peter R. Turner (Mathematics Department)	10
Dr. Richard Werking (Nimitz Library)	2
Naval Air Warfare Center	
Weapons Division	
China Lake, CA 93555-6001	
Head Research & Tech. Div. (NAWCWPNS-474000D)	2
Computational Sciences (NAWCWPNS-474400D)	2
Mary-Deirdre Coraggio (Library Division. C643)	2

Distribution List, cont.

Report No. NAWCADPAX- -96- 195 - TR

	No. of Copies
Naval Postgraduate School	
Monterey, CA 93943-5002	
Dudley Knox Library	2
Naval Research Laboratory(NRL)	
4555 Overlook Ave, SW	
Washington, DC 20375-5000	
Center for Computational Science (NRL-5590)	2
Superint., Lab. for Comput. Phy & Fluid Dynamics (NRL-6400)	2
Ruth H. Hooker Research Library (5220)	2
Naval Command, Control & Ocean Surveillance Center	
200 Catalina Blvd	
San Diego, CA 92147-5042	
Technical Library (NRAD-0274)	2
Signals Warfare Div (NRAD-77)	2
Analysis & Simulation Div. (NRAD-78)	2
Director of Navigation & Air C3 Dept. (NCCOSC-30)	2
Naval Air Warfare Center	
Aircraft Division Warminster	
Warminster, PA 18974-0591	
Warfare Planning Systems (4.5.2.1.00R07)	2
Tactical Inf. Systems (4.5.2.2.00R07)	2
Mission Comp. Processors (4.5.5.1.00R07)	2
Dr. Robert M. Williams (4.5.5.1.00R07)	20
Acoustic Sensors (4.5.5.4.00R07)	2
RF Sensors (4.5.5.5.00R07)	2
EO Sensors (4.5.5.6.00R07)	2
Inductive Analysis Branch (4.10.2.00R86)	2
TACAIR Analysis Division (4.10.1.00R86)	2
Operations Research Analysis Branch (4.10.1.00R86)	2
Advanced Concepts Branch (4.10.3.00R86)	2
Nav. Aval. Sys. Dev. Division (3.1.0.9)	2
Anthony Passamante (4.5.5.3.4.00R07)	2
Elect. Systems BR (4.8.2.2.00R08)	2
Dr. Richard Llorens (4.3.2.1.00R08)	2
Advanced Processors (4.5.5.1.00R07)	2
Mission & Sensors Integrations (4.5.5.3.000R07)	2
Applied Signal Process BR (4.5.5.3.4.00R07)	2